Reliable UDP Server

Implementation of the Header:

The header that I have designed is as below.

```
struct datagram_hdr{ int ack_flag;

unsigned long seq_num;

unsigned long ack_num;

char payload[1400];

}
```

Header

Ack_flag 32 bits
Seq_num 32 bits
Ack_num 32 bits
Payload 1400 bytes

Ack_flag: Indicates whether the datagram received is just an acknowledgement or it contains the data.

Ack flag={0,1}

Seq_num: This is a 32 bit unsigned long integer which indicates the sequence number of the UDP datagram.

Ack_num: This is the 32-bit acknowledgement number of the datagram which the client sends to the server as an indication of receipt of the correct datagram with correct sequence number on client side. It is equal to the sequence number received by the client plus one(plus one indicates that client received last one corectly and is requesting the next packet).

Payload: The data part of the header which carries the actual data in the UDP datagram. It is 1400 bytes.

My UDP packet size is **1424** bytes including the header.

Command line parameters:

Client: The client takes the hostname,port number of the server,filename to request for and window size in bytes as the parameter.

Server: The server takes the port number and the window size in bytes as the command line parameters.

Implementation of the Cumulative acknowledgements: The client which I have designed reads all the datagrams and sends back the cumulative acknowledgement of all the packets received in the window in sequence. It is assumed by the server if the cumulative acknowledgement is received that the client has received all the sequence number of packets before that acknowledgement number.

Implementation of the Sliding Window: I have implemented the sliding window concept by maintaining two variables called **sendbase** and **nextseqnum**. The former maintains the state of last sent packet which is not yet acknowledged by the client and the latter maintains the sequence number of the last packet sent to the client. As soon as an acknowledgement is received the sendbase moves forward by the last acknowledgement number received. And the nextseqnum is increased as per the number of packets being sent by the server.

Window size is fixed and is passed as command line parameters to both the client and the server. Since I am implementing header with structures, I am calculating the number of packets as the segment size by window size in bytes divided by the Maximum segment size.

Implementation of the Timeout interval ,Estimated RTT and Adaptive retransmission :

For estimating the timeout interval, I am calculating the sample RTT which is the time interval between the time at which first packet was sent by the server in the window and the time at which last acknowledgement was received by the server. Based on the **Jacobson** formula the weighted estimated RTT is calculated and **Deviation of estimated RTT** and **sample RTT** is calculated. The timeout interval is then calculated from the formula. The initial timeout interval which I took is 2 seconds on the server side. If the server receives three duplicate acknowledgements before it times out it retransmits the data which is not received and acknowledged by the client.

Implementation of the congestion control :TCP implements the congestion control by detecting packet loss in the network by two different ways. First is the detection of lost packets when duplicate acknowledgements are received from the client. Second is the timeout.

The server starts sending out the packets in slow start in which the congestion window is initially set to 1 and is increased by 1 MSS per acknowledgement received so it doubles every RTT. An initial ssthresh value that I have taken is 20000 bytes so that I can see when my server reaches both the **slow start** and the congestion avoidance mode. In **congestion avoidance** mode the window is increased by 1MSS per RTT and the ssthresh is set to congestion window divided by two. Congestion window is reset to 1.

If the congestion window reaches **ssthresh** the server goes in congestion avoidance.

When a timeout occurs the timeout interval is doubled in my program so as to not timeout again and again as the textbook suggests.

When timeout occurs also the ssthresh is set to half of the congestion window at that time and congestion window is reduced to 1.

Simulation of the lost packet :I have used rand() functions on the server side to drop few of the packets which the server is sending to the client. The client sends duplicate acknowledgements to indicate lost or out of order packets to the server and the packet is retransmitted by the server. I have taken different loss percentages and observed that if the lost percent is more then the server moves from slow start to the congestion avoidance faster. Also the timeout calculated by the Jacobson formula comes out to be too small and the server times out frequently.

Simulation of the server timeout: I have used the sleep() function on the client side to see if the server times out and retransmits the packets.The server does timeout according to the timeout interval calculated by the Jacobson algorithm and retransmits the packets. I have also implemented timeout on the client side so that the client does not keep on waiting in case the server doesn'r transmit the data which is less probable scenario.

Ideal simulation: When there is no packet loss the files get transferred reliably and faster.

It takes around 2 seconds and 30 microseconds to transfer a 1MB file without loss.

It takes more time to transfer the same file when I implement random packet drop as there are timeouts and duplicate acknowledgements to delay the file transfer process.